Freeform Search

Database: US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Term:

Display: 10 **Documents in Display Format:** - **Starting with Number** 1

Generate: ○ **Hit List** ◉ **Hit Count** ○ **Side by Side** ○ **Image**

Search Clear Interrupt

Search History

DATE: Monday, September 04, 2006 **Purge Queries** Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR			
L21	L18 and 705/35	108	L21
L20	L18 and 707.clas.	837	L20
L19	L18 and 705.clas.	819	L19
L18	l16 and (transaction with data or transaction near data or transaction adj data)	3990	L18
L17	L16 and 707/102	745	L17
L16	L15 amd (key with mask or key near mask or key adj mask)	41166	L16
L15	L14 amd (wildcard with key with element with value)	37730	L15
L14	L13 and (data with elements or data near elements or data adj elements)	12337	L14
L13	L11 and (key or column)and values and parameters	38172	L13
L12	L11 and (key or column)	71527	L12
L11	l1 and (relational or relation)	132086	L11
L10	705/30	1114	L10
L9	705/35	2602	L9
L8	705/1	6181	L8

L7	705.clas.	44326	L7
L6	707.clas.	37328	L6
L5	707/1	8378	L5
L4	707/102	8329	L4
L3	L2 and (wildcard with mask with field or wildcard near mask near field or wildcard adj mask adj field)	12	L3
L2	L1 and mask with field	3673	L2
L1	(database or data with base)	1030543	L1

END OF SEARCH HISTORY

L3: Entry 11 of 12 File: USPT Nov 30, 1999

DOCUMENT-IDENTIFIER: US 5995971 A

**** See image for Certificate of Correction ****

TITLE: Apparatus and accompanying methods, using a trie-indexed hierarchy forest, for storing wildcard-based patterns and, given an input key, retrieving, from the forest, a stored pattern that is identical to or more general than the key

Brief Summary Text (8):

Packet classification, for purposes of packet scheduling, will be performed, by constructing a so-called "key" from select fields, contained in a packet in order to associate the packet with a corresponding queue. These fields are illustratively source and destination addresses (typically IP addresses) and corresponding port designations (all of which are collectively referred to herein as "classification fields"). This association typically occurs by consistently concatenating the classification fields, in the packet being classified, into a key which, in turn, is used to access a data structure in order to retrieve therefrom an identification of a corresponding queue for that packet. Since a group of packets that have differing values for their classification fields can nevertheless be transmitted at the same rate and hence should be directed to the same queue, a mask field containing one or more so-called "wildcard" (oftentimes referred to as "don't care") values is often used, through logical combination with the classification fields, to yield an identification associated with a single queue. Generally speaking, this identification is viewed as a "classification pattern", i.e., a bit field having a length equal to the total length of the concatenated classification fields wherein any bit in the pattern can have a value of "1", "0" or "X", where "X" is a wildcard. As a result, a single pattern having a wildcard(s) therein can serve to classify an entire group of such packets. If a match occurs between the non-wildcard bits of the pattern (i.e., taking the wildcard value(s) into account) and corresponding bits of the classification fields for the packet being classified, then an associated queue designation for that pattern is accessed from the data structure.

Brief Summary Text (11):

While the art teaches various approaches for classifying packets, such as that typified in M. L. Bailey et al, "Pathfinder: A Pattern-Based Packet Classifier", Proceedings of First Symposium on Operating Systems Design and Implementation (OSDI), USENIX Assoc., Nov. 14-17 1994, pages 115-123 (hereinafter the "Bailey et al" paper), and W. Doeringer et al "Routing on Longest-matching Prefixes", IEEE/ACM Transactions on Networking, Vol. 4, No. 1, February 1996, pages 86-97 (hereinafter the "Doeringer et al" paper), these approaches, while efficient and effective in their target environments, are limited. In that regard, the techniques described therein exhibit retrieval times that are generally linearly related to the number of elements (n) in a classification database. A large network will have a substantial number of different patterns. Hence, the size of a classification data structure for such a network can be considerable which, in turn, will engender linearly increasing and hence relatively long retrieval times as the database expands. For packetized payload data that requires a high data rate, such retrieval times may be inordinately long and cause excessive delay to a recipient user or process.

Brief Summary Text (15):

Moreover, and apart from use in packet classification, a broad need continues to exist in the art for a generalized technique that, given specific input data (in the form of, e.g., a key), can rapidly retrieve a stored pattern, containing a wildcard(s) at any location therein, from a data structure--regardless of what the input data and patterns specifically represent. Such a technique would likely find widespread use including, but not limited, to packet classification. While the opposite problem of how to rapidly find a specific datum stored in a database given a generalized input pattern has been extensively investigated in the art, the converse problem, inherent in, e.g., a packet classifier, of how to swiftly retrieve a generalized stored pattern from a data structure given specific input data, e.g., classification fields, has apparently received scant attention in the art, which thus far, as discussed above, has yielded rather limited and disappointing results.

Brief Summary Text (19):
By virtue of our specific inventive teachings, given a specific input ("search") key, a stored pattern, that provides the most specific match, either identically or which completely subsumes the key, is retrieved from the stored database through a two-fold retrieval process. First, the search trie, having a topology of branch nodes, is traversed by branching, via child paths, at successive branch nodes therein as determined by corresponding bits in the key, each such bit being designated by a so-called "pivot bit" associated with each such branch node, until a lowest-level pattern node is reached. The pivot bit defines a bit position at which at least a pair of stored patterns in the rhizome disagree with each other. The branch nodes are organized in terms of increasing pivot bit value. Second, the pattern hierarchy, containing that lowest-level pattern node, is itself searched until the most specific pattern therein is found which matches or completely subsumes the key. A search ultimately succeeds if the retrieval process locates a stored pattern that most specifically matches, either identically or in terms of subsuming, the search key. If a search succeeds, then contents of a so-called "reference field" associated with the stored pattern are returned for subsequent use.

Brief Summary Text (20):
Our inventive rhizome exhibits the advantageous characteristic that the key will match either a specific pattern stored in the rhizome or an increasingly general pattern stored above it in a hierarchical chain, or that key will not have a matching pattern stored anywhere in the rhizome. Hence, our inventive rhizome effectively narrows down a search for a pattern that most specifically matches a key from an entire database of stored patterns to just a linear chain of patterns in order of strict hierarchy, the latter being usually substantially smaller in size than the former. Specifically, through our inventive rhizome, the maximum number of bit comparisons that needs to be made to retrieve a pattern for a given search key generally equals the number of bits in the key itself plus the number of patterns, in the longest hierarchy chain in the hierarchy forest, less one. Moreover, owing to the strict hierarchy inherent in every hierarchical chain in our inventive rhizome, i.e., where patterns at higher levels in a given pattern hierarchy completely subsume those at lower levels therein, then, if a bit-by-bit comparison is to be made for patterns in a pattern hierarchical chain against an input key, then as increasingly general patterns are encountered in that hierarchy, there is no need to compare a bit in any bit position of the key that has previously been compared against and found to match that in any lower level pattern in that chain. This, in turn, further reduces processing time required to find a desired pattern in the rhizome.

Brief Summary Text (22):
Our inventive rhizome provides an advantageous feature that, for large databases, asymptotically, average search delay through the rhizome is a logarithmic function of a number of patterns stored in the rhizome rather than linear in the number (n) of patterns, as occurs with conventional databases of patterns containing

wildcards. Hence, use of our inventive rhizome, particularly with a large and increasing classification database--as frequently occurs in packet classifiers, appears to provide significant time savings over conventional packet classification techniques.

Drawing Description Text (4):
FIG. 2A and 2B depict typical key 200, and pattern 250 and its accompanying mask field 255, respectively, used in classifying a packet for scheduling;

Drawing Description Text (9):
FIGS. 6A and 6B graphically depict a conventional database retrieval problem and an inverse problem inherent in a packet classifier, respectively;

Drawing Description Text (21):
FIG. 12B depicts sub-structure portion 1002' which results after a node has been inserted into sub-structure 1002 shown in FIG. 12A, along with corresponding contents of VALUE, MASK and IMASK fields of each of the nodes shown in sub-structure portion 1002' both before and after the insertion;

Detailed Description Text (9):
In particular, each packet contains source and destination IP (Internet Protocol) addresses, i.e., an network address of a computer or other network device ("source device") at which that packet was created and a network address of a computer or other network device ("destination device") at which that packet is to be delivered. The packet also contains designations of source and destination ports, i.e., specific ports at the source and destination devices at which the packet originated and at which the packet is to be received, respectively. The source and destination addresses, and source and destination port designations collectively form the "classification fields". In other implementations, the classification fields may contain: other fields from a packet in addition to these addresses and port designations, fewer fields than these addresses and port designations, or even other fields from a packet. In any event, for purposes of the present embodiment, these addresses and port designations, when appropriately concatenated to form key 200, are used to access a so-called "classification pattern" from a stored database. As shown, key 200 contains four successive contiguous fields as the classification fields: source port designation field 210, destination port designation field 220, source address field 230 and destination address field 240. Though classification patterns are a specific type of pattern, for brevity, we will use these two terms synonymously during the course of describing our invention in the context of a packet classifier.

Detailed Description Text (10):
For each different transmission queue established at a computer, a separate corresponding pattern exists and is stored within a database. Moreover, multiple stored patterns may often correspond to the same queue. In any event, complications arise inasmuch as packets with different values for their classification fields are often to be transmitted at a common data rate, and hence are to be directed to a common transmission queue. In order to direct packets with different values of the classification fields to a common queue, each pattern, such as pattern 250 shown in FIG. 2B, has mask field 255 associated therewith, wherein a separate mask sub-field is provided for each of the individual classification fields, i.e., for each of the source and destination address fields and the source and destination port designation fields. As shown, pattern 250 is formed of: source port designation field 264, destination port designation field 274, source address field 284 and destination address field 294; with mask field 255 being formed, with corresponding concatenated sub-fields, of: source port designation mask sub-field 267, destination port mask sub-field 277, source address mask sub-field 287 and destination address mask sub-field 297.

Detailed Description Text (11):

Each mask sub-field identifies, by a one in a given bit position, each bit that is important (i.e., must be matched in value) in a corresponding classification field, (i.e., an address or port designation), and, by a zero in a given bit position, which bit in that classification field can be ignored, i.e., that which is a so-called "don't care" bit or (as will be equivalently referred to hereinafter) "wildcard". For an example shown in FIG. 2B, consider, as example 285, an address in source address field 284 to be the value 101001010 and a corresponding mask in mask sub-field 287 to be the value 111001110. What this means is that the source address bit-values in address bit positions (starting at MSB (most significant bit) position zero on the left and counting to the right) three, four and eight can be ignored. Hence, the source address becomes 101XX101X where "X" represents a "wildcard". Hence, any packet having a source address that contains the values "101" in bit positions zero through two and again in bit positions five through seven, regardless of its bit values in bit positions three, four and eight, will match the source address in pattern 250. By virtue of having a wildcard(s), a single source address in a pattern, such as that in pattern 250, can match against multiple different source addresses in packets. To specify just one particular source address rather than a group, all bits in the mask sub-field for the source address in a pattern would be set to one, i.e., denoting no wildcards; hence requiring a match over all bits in the address field with a corresponding source address in a key for a packet. The source port designation, destination port designation and destination address are all identically specified for a pattern through by a combination of an address or port designation in fields 264, 274 and 294, respectively, with a corresponding value in mask sub-fields 267, 277 and 297, respectively, of mask field 255.

Detailed Description Text (12):
As noted above, once the database search successfully retrieves a pattern that matches or subsumes within it the key for a packet, the queue designation associated with that pattern specifies a particular transmission queue, within the scheduler, to which that packet will be directed. The queue designations are simply predefined and stored in the database. Since the manner through which the bits in the queue designations are set to specify an associated transmission queue is irrelevant to the present invention, we will omit all those details from the ensuing discussion.

Detailed Description Text (17):
As shown in FIG. 3, incoming information can arise from two illustrative external sources: network supplied information, e.g., from the Internet and/or other networked facility such as an intra-net (all generally shown as network 60 in FIG. 1), through network connection 50 to communications interface 330 (shown in FIG. 3), or from a dedicated input source via path(s) 305 to input interfaces 310. Dedicated input can originate from a wide variety of sources, e.g., an external database, video feed, scanner or other input source. Input interfaces 310 are connected to path(s) 305 and contain appropriate circuitry to provide the necessary and corresponding electrical connections required to physically connect and interface each differing dedicated source of input information to computer system 100. Under control of the operating system, application programs 130 exchange commands and data with the external sources, via network connection 50 or path(s) 305, to transmit and receive information typically requested by a user during program execution.

Detailed Description Text (24):
Upon entry to this process--which is separately performed for each packet, block 555 is executed which extracts the fields of interest, i.e., the classification fields, from a header of that packet. Thereafter, the extracted fields are consistently concatenated, by block 560, into a single key, i.e., a current search key, to form key 220 shown in FIG. 2. To minimize search delay, recently used keys and their associated flows are cached within memory 340 (see FIG. 3) so as to advantageously eliminate the need to conduct a complete database retrieval

operation on each and every search key. Hence, once the current search key is formed, block 565, as shown in FIG. 5B, examines a cache to locate that key. Next, decision block 570 determines whether the cache contains the current search key. If it does, then execution proceeds, via YES path 572, to block 575 which retrieves the pattern, and its associated queue designation, for this packet from the cache. Alternatively, if the cache does not contain the current search key, then this decision block routes execution, via NO path 574, to block 580. This latter block, when executed, retrieves the pattern for the current block by searching our inventive data structure, also referred to herein as a "rhizome" and which will be discussed in considerable detail below, for that pattern. Once this pattern is located, either by retrieving it from the cache or from our inventive rhizome, the queue designation therefor is then returned, through execution of block 590, to a requesting client, thereby completing execution of process 550 for the packet. Though not specifically shown, once the pattern and the queue designation are retrieved from our inventive rhizome, these items are then cached for subsequent short-term retrieval.

Detailed Description Text (25):
B. Conventional Database Searching vis-a-vis Packet Classification Searching

Detailed Description Text (26):
Our inventive data structure and retrieval algorithms, particularly for use in packet classification, are directed to an entirely opposite problem to that routinely encountered with conventional databases.

Detailed Description Text (27):
In particular, and as shown in FIG. 6A, a conventional database, illustrated by database 630, contains specific pieces of information, such as entries 630.sub.1, 630.sub.2, 630.sub.3 and 630.sub.4, each of which stores a different numeric value. These values are illustratively 1011, 1010, 1111 and 0110 for entries 630.sub.1, 630.sub.2, 630.sub.3 and 630.sub.4, respectively. With such a conventional database, a generalized input pattern, such as pattern 610 here having an illustrative value 1X11 with a wildcard in bit position one, is applied; the database manager is instructed to return all stored data values that match, i.e., are identical to or are subsumed by, the pattern. Accordingly, entries 630.sub.1, and 630.sub.3 will be returned as containing values 1011 and 1111 that are both subsumed within input pattern 1X11.

Detailed Description Text (28):
In sharp contrast, we are faced with the problem, inherent in a packet classifier and as illustrated in FIG. 6B, of searching a database of generalized patterns for a classification pattern that matches, either identically or subsumes within it, a specific input value, i.e., a key. Here, database 670 illustratively contains stored values "1XX10" in entry 670.sub.1, "0X101" in entry 670.sub.2, "01101" in entry 670.sub.3 and "01001" in entry 670.sub.4. Illustrative input key 650, containing a value "10110", is applied to a retrieval process in an attempt to return a stored classification pattern that is either identical to or contains the key. Here, the retrieval succeeds in returning the stored classification pattern "1XX10" in entry 670.sub.1. As one can appreciate, the stored and returned classification pattern "1XX10", containing wildcards in bit positions one and two, subsumes within it the input key "10110".

Detailed Description Text (29):
As noted, a specific input key can identically match or be subsumed within a generalized stored classification pattern. The number of different keys that will be subsumed within any given classification pattern will be dictated by the number of wildcards contained within that pattern. If a stored classification pattern contains no wildcards, then only one key, i.e., that which identically matches the stored classification pattern, will result in that pattern being returned. Alternatively, if a stored four-bit classification pattern contains all wildcards,

i.e., the value "XXXX", then any possible four-bit key, for a total of 16 different four-bit keys, will result in that stored classification pattern being returned. In that regard, consider FIG. 7A which depicts stored eight-bit classification pattern 710 having three wildcards, in bit positions two, three and seven. As a result, eight different eight-bit keys 750, as shown in FIG. 7B, when applied to a database containing stored classification pattern 710 will result in that particular classification pattern being returned.

Detailed Description Text (31):
A pattern classifier functions by retrieving, for a given specific input key, the most specifically matching stored classification pattern from among all the classification patterns stored in a pattern hierarchy forest in a database.

Detailed Description Text (76):
As with a conventional Patricia tree, a search of our inventive rhizome is conducted on the basis of an input search key. In a conventional Patricia tree as discussed above, either the search key will match a value found by tracing through the tree or that key will not have a matching value anywhere in the tree. With our inventive rhizome, the search key will match either a specific pattern stored in the rhizome or a increasingly general pattern stored above it in a hierarchical chain, or that key will not have a matching pattern stored anywhere in the rhizome. Hence, the binary search trie effectively narrows down a search from an entire database of stored patterns to just a linear chain of patterns in order of strict hierarchy, the latter being usually substantially smaller in size than the former. A search succeeds if the retrieval process locates a stored pattern that most specifically matches, either identically or in terms of subsuming, the search key.

Detailed Description Text (86):
In particular, data structure 1300 shown in FIG. 13 contains data 1310 which is stored at a branch node, and data 1350 which is stored at each pattern node. Data for VALUE, MASK and IMASK fields 1315, 1320 and 1325, are stored for each node, regardless of whether that node is a branch or pattern node. The MASK and IMASK fields, which will shortly be defined and discussed in considerable detail below, are necessary to specify a wildcard(s) at a given pattern node and to propagate the existence of that wildcard(s), at that node, downward into underlying structure of the rhizome. The reason, as will be described in detail below, supporting the need for such propagation is to provide a mechanism at each branch node that indicates whether a section of the structure, starting at that node and extending upward, needs to be replicated (i.e., to provide alternate search paths) during insertion. This propagated information is also necessary, as will be discussed below, for determining, during removal, just how much structure in the rhizome is redundant, in view of a pattern to be removed.

Detailed Description Text (90):
Now, as to the VALUE, MASK and IMASK (being an abbreviation for an "inverse mask") fields, the purpose of each of these fields differs between a pattern node and a branch node. To facilitate understanding, we will discuss each of these fields, first in terms of its use in a pattern node and then in terms of its use in a branch node.

Detailed Description Text (92):
With respect to the MASK field used in any pattern node, this field specifies the existence of a wildcard in the corresponding pattern (stored in the associated VALUE field). In that regard, if a given bit in the MASK field is a zero, then the corresponding bit in a pattern is a wildcard (denoted by an "X" in the patterns shown, for example, in FIG. 10); if the given bit in the MASK field is a one, then the corresponding bit of the pattern equals the corresponding bit in the VALUE field. Thus, the pattern 00101100XX001011 (stored in pattern node 1031 shown in FIG. 10) is represented by a MASK field of 1111111100111111 and a VALUE field 0010110000001011. The two adjacent bits in the eighth and ninth bit positions in

this pattern, given the wildcards situated there, are irrelevant and could equally be 00, 01, 01 or 11. However, to simplify the logic and testing used in our inventive insertion routine for our inventive rhizome, as discussed in detail below in conjunction with FIGS. 15, 16A and 16B, we set any bit position in the VALUE field, at which a wildcard exists at that bit position in the pattern, to zero. For a shorthand notation, though each of the pattern nodes situated in rhizome 1000 shown in FIG. 10 is formed of a combination of the VALUE and MASK fields, for this and other figures, a wildcard at any bit position in the VALUE field (identified by a corresponding zero bit in MASK field) is set to "X" to denote that wildcard. As shown in FIG. 2B and discussed above in conjunction therewith, a pattern, for use in packet scheduling, such as pattern 250, is formed by consistently concatenating the classification fields; a MASK field therefor, such as mask field 255, is formed by consistently concatenating the corresponding mask sub-fields for each of the individual classification fields.

Detailed Description Text (93):
As another example, consider the pattern 00101100XX0XXX11 stored in pattern node 1041 shown in FIG. 10, this pattern is represented by a VALUE field of 0010110000000011 and a MASK field of 1111111100100011. Likewise, consider the pattern 0010XX00XXXXXX11 stored in pattern node 1045. This pattern is represented by a VALUE field of 0010000000000011 and a MASK field of 1111001100000011.

Detailed Description Text (94):
For each pattern node, the IMASK field of that node is set equal to the IMASK field of its godparent, if it exists, of that node. For any pattern node that does not have a godparent, the IMASK field of that particular pattern node is set equal to its own MASK field. Inasmuch as pattern hierarchies in any hierarchy chain follow strict hierarchical rules as the chain is incrementally traversed upward to successively more general patterns, then, setting the IMASK field of a pattern node to that of its godparent is equivalent to stating that the IMASK field of that pattern node is equal to a conjunction (i.e., logical AND combination) of the MASK field of that node and the IMASK field of its godparent. This occurs because, as a result of the strict hierarchy in a pattern chain, the MASK field of the godparent is guaranteed not to have any bits set therein that are not set in the MASK field of the pattern node immediately below it (i.e., a godchild) in the chain.

Detailed Description Text (95):
Furthermore, in each instance where hierarchy is irrelevant or nonexistent, such as for a hierarchy chain that contains just one pattern node, then the MASK and IMASK fields of that pattern node are set equal to each other.

Detailed Description Text (96):
As noted above, the MASK, IMASK and VALUE fields serve a different purpose for any branch node than for any pattern node. Specifically, the fields of any given branch node specify, in condensed form, information about the fields of all patterns reachable from that branch node. There are five possible states maintained for each bit position: (a) all descendent patterns require that the bit be zero; (b) all descendent patterns require that the bit be one: (c) some but not all descendent patterns require that the bit be zero; (d) some but not all descendent patterns require that the bit be one; and (e) no descendent patterns care about the value of the bit. This information is necessary, as stated above, during: (a) insertion, for determining just how what portion of the rhizome needs to be replicated in order to insert a new pattern, and (b) removal, for determining, just how much structure in the rhizome is redundant, in view of a pattern to be removed.

Detailed Description Text (97):
In particular, the MASK field for any branch node is set to the disjunction (logical OR combination) of the MASK fields of its two child nodes, up to the bit position specified by the pivot bit of this branch node. Since this is a recursive definition extending upward through both the search and hierarchy structures, a bit

value of one in a MASK field of a branch node indicates that at least one pattern node, reachable from (i.e., a lineal descendant of) that branch node, cares about the value of that bit. If a bit position in a MASK field of a branch node is a zero, then this means that no descendent pattern, i.e., any of those in a pattern hierarchy chain reachable from this branch node, cares about this bit position; hence, the value at this bit position is irrelevant to all those patterns.

Detailed Description Text (110):
To further enhance understanding, we will now turn to FIGS. 12A and 12B. These figures diagrammatically depict an insertion operation in our inventive rhizome and the resulting impact on the VALUE, MASK and IMASK fields. For ease of understanding, the reader should simultaneously refer to both of these figures during the course of the ensuing discussion.

Detailed Description Text (111):
Specifically and illustratively, a new pattern 100X1011XX111X00 is to be inserted into rhizome 1000 shown in FIG. 10. To simplify the drawings, FIG. 12A depicts a vertical chain in this rhizome that will be affected by this insertion. As can be seen from comparing this new pattern with those already stored in the rhizome, the lowest bit position at which a bit disagreement occurs is in bit position ten. Hence, a new branch node, with a pivot bit of ten, specifically node 1210, will be added to sub-structure 1002 shown in FIG. 12A. This new pattern will be stored in a new pattern node, specifically pattern node 1220. To provide proper links to this new pattern node, an existing link from a child one branch emanating from node 1080 to node 1085, this link indicated by a dotted line in FIG. 12B, will be removed and replaced by a link to new branch node 1210. The child zero branch of branch node 1210 will link to node 1085; the child one branch of new branch node 1210 will link to new pattern node 1220, thereby forming sub-structure, here vertical chain, 1002'. The new links inserted into sub-structure 1002 to accommodate the new pattern are shown by solid double lines. In addition, during node insertion, the contents of the VALUE, MASK and IMASK fields for those nodes in the chain below new branch node 1210 are all updated accordingly, through disjunction or conjunction as discussed above, to reflect the corresponding values for the new pattern node, particularly the wildcards therein. In that regard, to the right of each of nodes 1070, 1075, 1080, 1210, 1085, 1090, 1092 and 1220 in FIG. 12B are corresponding tables 1070.sub.t, 1075.sub.t, 1080.sub.t, 1210.sub.t, 1085.sub.t, 1090.sub.t, 1092.sub.t and 1220.sub.t. Each of these tables specifies the contents of the VALUE, MASK and IMASK fields of that corresponding node both without the new node (status being "without"), i.e., for sub-structure 1002 in FIG. 12A, and with the new node having been inserted into the rhizome (status being "with"), i.e., for sub-structure 1002' in FIG. 12B.

Detailed Description Text (112):
Though the child one pointer would also be updated accordingly for node 1080, for simplification, all fields other than VALUE, MASK and IMASK for each of the nodes in sub-structure 1002' have been omitted from FIG. 12B. Based on the description, as given above, of each of these other fields, it will be quite apparent to those skilled in the art how the contents of those fields would be set or modified, as required, to accommodate insertion (as well as for removal as will be described immediately below).

Detailed Description Text (114):
Essentially, removal involves deleting desired nodal structure, specifically desired leaf nodes and associated branch nodes that provide paths to these leaf nodes, and modifying the VALUE, MASK and IMASK fields and associated child pointers of remaining affected nodes in the rhizome to eliminate the deleted structure. For every stored pattern that is to removed from our inventive rhizome, one or more branch nodes, depending upon any resulting redundancy in the structure occasioned by the removal, will be eliminated. Whether a branch node is to be removed or not is determined by whether the pattern being removed cares about the pivot bit of

that branch node. If it cares, but no other hierarchically related pattern so cares, then that branch node is removed. Owing to wildcards, removal of a pattern node can engender removing a significant part of the search trie--not just one branch node and an associated path as shown in FIG. 9F for a conventional Patricia tree.

Detailed Description Text (115):
To illustrate removal consider FIG. 12C which depicts sub-structure 1002'. Pattern node 1220, which has been inserted into sub-structure 1002 shown in FIG. 12A to produce sub-structure 1002' shown in FIGS. 12B and 12C, will now be removed. Since this node is the only node that presents a bit disagreement at bit position ten, that node along with associated branch node 1210 are both removed. To do so, these nodes are simply deleted and the values of fields VALUE, MASK and IMASK are updated accordingly. The values for these fields, prior to removal, are indicated in the tables shown in FIG. 12B and specifically those under the "with" status designation. With the removal of these nodes, the resulting updated fields will contain the values shown in these tables, specifically those under the "without" designation.

Detailed Description Text (119):
FIGS. 12D, 12E and 12F collectively depict an example of insertion that necessitates replication. Here, FIG. 12D depicts sub-structure portion 1004 of rhizome 1000 shown in FIG. 10. The nodes shown in FIG. 12D are re-arranged in FIG. 12E to provide room in the figure for newly added structure, to be shown in FIG. 12E. Here, a new pattern, specifically 0011110010011110, is to be inserted into sub-structure portion 1004. Once inserted, sub-structure portion 1004' depicted in FIG. 12F results. The new pattern, stored in pattern node 1230, causes the creation of new branch node 1240 with a pivot bit of four. This occurs inasmuch as the first bit at which a disagreement now occurs, between any existing pattern stored in the structure and the new pattern, is at bit position four. Since existing patterns 0011010001110100 and 00X101X00X110100 that disagree with the new pattern have a zero in bit position four, the existing sub-structure for these particular patterns must reside below child zero of the new branch node. Moreover, since two of the patterns reachable through new branch node 1240, specifically 00XXX00111010010 and 0011X10000110000, have a wildcard in bit position four, these patterns must be reachable from both sides of this new branch node. Therefore, structure leading to the pattern nodes for these two patterns must be replicated below child one of the new branch node. Inasmuch as the first bit at which these two particular patterns disagree amongst themselves occurs at bit position five, then branch node 1050 is replicated as branch node 1245 to distinguish between these two patterns. Finally, a new branch node 1250 with a pivot bit of eight is created to completely distinguish the newly inserted pattern from those already in the structure. The replicated structure, here pattern nodes 1018 and 1065 contained in dashed blocks, is achieved by additional child zero paths 1262 and 1266 emanating from new branch nodes 1245 and 1250, respectively. To greatly simplify FIGS. 12D-12F, the values of the VALUE, MASK and IMASK fields have been intentionally omitted, for each of the nodes, therefrom; nevertheless, these values can be easily determined through the specific manner set forth above.

Detailed Description Text (124):
There may exist applications for which it is necessary to store patterns containing wildcards in a database, yet for which no need for hierarchical pattern retrieval is necessary. In such cases, virtually all of the structure thus far described is still required, with the exception of the godparent pointer 1365 in nodal data structure 1300, which can be omitted. However, the routines which implement retrieval, insertion, and removal can be simplified significantly for such applications. Therefore, the present section will describe two embodiments of the invention, the hierarchical rhizome and the non-hierarchical rhizome, and the routines will similarly be known as hierarchical retrieval and non-hierarchical retrieval, and so forth. To simplify reader understanding, the description refers

to the hierarchical rhizome, in keeping with the description thus far presented. Where appropriate, the ensuing description will specifically delineate those portions of the routines which can be deleted for the non-hierarchical embodiment.

Detailed Description Text (138):
In particular, routine 1500 is entered with two parameters: VALUE and MASK. Together, as noted above, these parameters collectively specify the new pattern to be inserted, including its wildcard(s), if any. Upon entry into routine 1500, execution first proceeds to block 1510. This block, when executed, allocates sufficient memory area to accommodate a new pattern node, i.e., sufficient memory locations to store data structure 1300 (as shown in FIG. 13) for this new node. Once this allocation has occurred, block 1510 then sets the values of various fields in this structure appropriately for this new node. In particular, in order to insure that the bits of the VALUE field that correspond to wildcard indications in the MASK field are zero, the VALUE field is set to the conjunction of the given VALUE and MASK values supplied, upon entry, to this routine. The MASK and IMASK fields are set equal to the MASK value supplied to this routine. The pivot bit for this new node is set equal to the number of bits in this pattern (thereby, as in the manner stated above, distinguishing this node from a branch node). Thereafter, the pointer to godparent field for this new pattern node is set equal to a null value. Finally, the reference field is then appropriately set to a particular predefined data value, e.g., a queue designation, given for this node. Since the manner through which the value for the reference field is determined and assigned is immaterial to the present invention, it will not be described in any further detail.

Detailed Description Text (146):
If the pointed-to node is a branch node, then this routine will update the fields (specifically VALUE, MASK and IMASK) for the pointed-to node, and will recursively call itself. Specifically, if the new pattern does not care about the value of the bit specified by the pivot bit of the pointed-to node--i.e., a wildcard exists in new pattern at this bit position, then this routine recurses itself on both children of this node. Otherwise, if the new pattern does not have a wildcard at the bit position indexed by the pivot bit of the pointed-to node, then routine 1600 only recurses itself on the child determined by the bit at this position in the new pattern.

Detailed Description Text (149):
If the bit index is less than the pivot bit of the pointed-to node, then decision block 1606 routes execution, via YES path 1607, to decision block 1610. The latter decision block determines, by examining corresponding bits, whether a bit, specified by the bit index, in the new pattern and that in the pointed-to node disagree, meaning that one pattern specifies that the bit be zero and the other pattern specifies that it be one. If no such disagreement exists, then execution is fed back from decision block 1610, via NO path 1612, to block 1638, which, when executed, increments the bit index by one to point to the next bit. Execution then proceeds to decision block 1606 to test this next bit, and so forth. Alternatively, if such a disagreement exists, then decision block 1610 routes execution, via YES path 1614, to block 1615. Consequently, a new branch node must now be inserted into the rhizome, specifically the search trie therein, with a pivot bit that specifies the bit position where the disagreement exists. To do so, block 1615 allocates a new branch node and appropriately sets various fields associated with this node. In particular, the VALUE field is set to the disjunction (logical OR combination) of the VALUE field of the new pattern and the VALUE field of the pointed-to node. Similarly, the MASK field is set to the disjunction of the MASK field of the new pattern and the MASK field of the pointed-to node. Also, the IMASK field is set to the conjunction (logical AND combination) of the MASK field of the new pattern and the IMASK field of the pointed-to node. By setting these fields appropriately, based on those of higher-level nodes in the rhizome, the existence of wildcards at pattern nodes reachable through the new branch node will be propagated to that

branch node. In addition, block 1615 sets a specific child pointer of the new branch node to the pointed-to node. The specific child pointer, i.e., the child zero or child one pointer, is selected based on the indexed bit of the VALUE field of the pointed-to node. The pivot bit for the new branch node is set equal to the current bit index. Once these operations have completed, execution proceeds from block 1615 to decision block 1620. This latter decision block determines whether a wildcard exists, in patterns reachable by the new branch node, at the bit position specified by the current bit index. If no such wildcards exist at this bit position, i.e., the value of the IMASK bit at this bit position in the pointed-to node is one, then decision block 1620 routes execution, via YES path 1624, to block 1625. This latter block sets the specific child pointer for the new branch node, as designated by the indexed bit of the VALUE field of the new pattern, to point to the new pattern node. Once this occurs, execution then exits from routine 1600. Alternatively, if a wildcard exists, in patterns reachable by the new branch node, at the bit position specified by the current bit index, then decision block 1620 routes execution, via NO path 1622, to block 1630. As a result of the wildcard, portions of the existing rhizome, above the pointed-to node, must be replicated to properly support the wildcard. Consequently, block 1630, when executed, invokes Replicate routine 1900 (as will be discussed in detail below in conjunction with FIG. 19) with two arguments: POINTED-TO NODE and the current INDEXED BIT. The replica returned by the replicate routine becomes a child of the new branch node, specifically the child corresponding to the indexed bit of the new pattern. The value of POINTER is set to point to this child. Once block 1630 has fully executed, i.e., Replicate routine 1900 has completed all its recursion and POINTER has been finally set, then execution is fed back, via path 1635, to block 1638 to increment the bit index for examining the next bit position in the new pattern vis-a-vis the next pivot bit in the rhizome, and so forth.

Detailed Description Text (150):
When the bit index reaches the value of the pivot bit of the pointed-to node, execution then proceeds, via NO path 1609 emanating from decision block 1606, to decision block 1640. This latter decision block determines, based on testing the value of the pivot bit stored for the new node, whether that node is a pattern node or not. If the pointed-to node is a branch node, then, having traversed this far up the search trie to where the index bit equals the pivot bit of the pointed-to node, then the pattern needs to be inserted somewhere above the pointed-to node. Therefore, execution proceeds via NO path 1641 to block 1648, where the VALUE, MASK AND IMASK fields in the pointed-to node are modified accordingly to reflect this ancestry. In particular, the VALUE and MASK fields of the pointed-to node are formed by separately disjoining the existing VALUE and MASK fields of this node with the VALUE and MASK fields of the new pattern, respectively. The IMASK field of the pointed-to node is formed by conjoining the existing IMASK field of this node with the IMASK field of the new pattern. Once these fields for the pointed-to node have been updated, execution proceeds to decision block 1650 to determine whether a wildcard exists in the new pattern at the bit position indexed by the pivot bit of the pointed-to node. If a wildcard does not exist, as given by a one in the MASK field at this bit position in the new pattern, then the new branch node will be inserted (as well as its supporting structure, with replication if necessary) on the child zero path from the pointed-to node. In this case, execution proceeds, via YES path 1651 emanating from decision block 1650, to block 1658. This latter block recursively invokes the present routine, i.e., Node Insert routine 1600, on a child of the pointed-to node, that child being selected by a bit of the new pattern indexed by the pivot bit of the pointed-to node. Once recursion fully completes and block 1658 has completely executed, execution then exits from routine 1600. Alternatively, if a wildcard exists, as given by a zero in the MASK field at this bit position in the new pattern, then nodes will need to be inserted on both sides (i.e., on both child branches) of the pointed-to node, with appropriate replication of supporting rhizome structure to support each such new node. Consequently, execution proceeds, via NO path 1653, emanating from decision block 1650, to block 1655. This latter block recursively invokes Node Insert routine 1600 for child one

of the pointed-to node. Thereafter, once block 1655 has fully executed, execution next proceeds to block 1658 to appropriately insert nodes, again recursively, on now the child zero of the pointed-to node (remembering, as noted above, that a zero is set in a bit position in the VALUE field for a pattern node that has a wildcard at that position), after which execution exits from routine 1600.

Detailed Description Text (159):
Specifically, upon entry into routine 1900, execution first proceeds to block 1905. This block, when executed, determines whether the pointed-to node is a pattern node. In the non-hierarchical replicate routine, the blocks within dotted box 1930, specifically decision block 1932 and block 1938, can be omitted. Thus, in this case, if the pointed-to node is a pattern node, then execution exits via YES path 1909 and dashed path 1910, with the address of the current pattern node being returned. In the hierarchical replicate routine, if the pointed-to node is a pattern node, then execution proceeds, via YES path 1919, to decision block 1932. This decision block determines whether the bit, indexed by the pivot bit, in the MASK field of the pointed-to node equals one. If this bit does not equal one, i.e., it equals zero, hence specifying a bit for which a wildcard exists either in this pattern node or higher in the pattern hierarchy and for which this pattern does not care, then execution exits routine 1900, via NO path 1936 emanating from decision block 1932. The address of the pointed-to node is returned. Alternatively, if this bit equals one, i.e., this pattern cares about the value at this bit position, then decision block 1932 routes execution, via YES path 1934, to block 1938. This latter block recursively invokes the present routine, i.e., Replicate routine 1900, but with POINTER set to specify the godparent of the current pattern node. After this recursive execution, execution exits from this routine with an address of a god-ancestor being returned.

Detailed Description Text (163):
In particular, decision blocks 1965 and 1975 collectively determine whether a bit, indexed by the current pivot bit in the pointed-to node, in the MASK field, of head nodes of both replicated branches of the current node, equals one, i.e., whether patterns reachable through both of these child branches care about the value of the pivot bit of the current node. To accomplish this, decision block 1965 first tests the replicated child zero branch of the current node. If the indexed bit in the MASK field of the head node of the replicated child zero branch equals zero, i.e., indicative of a bit that all patterns reachable through this node do not care about (a wildcard exists at that bit position in all of these patterns), then decision block 1965 routes execution, via NO path 1969, to block 1970. This latter block, when executed, invokes Eliminate routine 2000 (discussed in detail below in conjunction with FIG. 20) to remove this particular replicated nodal structure. Once block 1970 completes execution, execution exits from routine 1900 and the address of the replicate of the child one branch is returned. Alternatively, if indexed bit in the MASK field of the head node of the replicated child zero branch equals one, then decision block 1965 routes execution, via YES path 1967, to decision block 1975. This latter decision block tests the child one branch of the replicate node. If the indexed bit in the MASK field of the head node of the replicated child one branch equals zero, i.e., indicative of a bit that all patterns reachable through this node do not care about (a wildcard exists at that bit position in all of these patterns), then decision block 1965 routes execution, via NO path 1979, to block 1980. This latter block, when executed, invokes Eliminate routine 2000 to remove this particular replicated nodal structure. Once block 1980 completes execution, execution exits from routine 1900 and the address of the replicate of the child zero branch is returned. Alternatively, if indexed bit in the MASK field of the head node of the replicated child one branch equals one, then decision block 1975 routes execution, via YES path 1977, to block 1985. This latter block, when executed, replicates the current node by allocating a new branch node and setting the contents of its fields. Specifically, the VALUE and MASK fields for the new branch node are set to the disjunction of the VALUE and MASK fields, respectively, of the replicate head nodes. The IMASK field for this

new node is set to the conjunction of the IMASK field of the replicate head nodes. The child pointers are set to point to the respective replicas of the child branches of the current node. Lastly, the pivot bit of this new node is set to the current pivot bit of the pointed-to node. After these operations have occurred, execution then exits from routine 1900 with the address of the new node being returned.

Detailed Description Text (175):
In the hierarchical node remove routine, if the current node is a pattern node, then execution proceeds, via YES path 1803, to decision block 1855. This decision block, when executed, determines whether the pointed-to node stores the particular pattern that is to be removed, i.e., whether this stored pattern identically matches that which is to be removed. If it does match, then decision block 1855 routes execution, via YES path 1859, to block 1860 to remove this particular node. If, however, the pattern to be removed does not exactly match that at the pointed-to node, then decision block 1855 routes execution, via NO path 1857, to decision block 1865. This latter decision block tests whether the pointed-to node has a godparent. If the pointed-to node has a godparent, then decision block 1865 routes execution, via YES path 1869, to block 1870. This latter block recursively invokes this routine, i.e., Node Remove routine 1800, on the godparent of the pointed-to node, i.e., POINTER is set to point to this godparent. Once all such recursive execution has completed, execution of block 1870 completes with execution proceeding to decision block 1875. Execution also reaches this decision block, via NO path 1867 emanating from decision block 1865, in the event the pointed-to node does not have a godparent. Decision block 1875 again tests whether the pointed-to node has a godparent, since this godparent may have been just been removed through now completed recursive execution of routine 1800 (invoked through block 1870). In the event the pointed-to node still has a godparent, i.e., the current node previously had more than one godparent, only one of which was removed via recursive execution of routine 1800, then decision block 1875 routes execution, via YES path 1877, to block 1880. This latter block, when executed, sets the IMASK field in the pointed-to node equal to the IMASK field of its godparent. Thereafter, execution exits from routine 1800. Alternatively, if the pointed-to node now does not have a godparent, i.e., this node is now the highest pattern node in a pattern hierarchy chain, then decision block 1875 routes execution, via NO path 1879, to block 1885. This latter block, when executed, sets the IMASK field in the pointed-to node equal to the MASK field for this node. Thereafter, execution exits from routine 1800.

Detailed Description Text (176):
If, on the other hand, decision block 1802 determines that the pointed-to node is a branch node, then block 1802 routes execution, via NO path 1804, to Branch Node Removal sub-routine 1810. Upon entry into sub-routine 1810, execution first proceeds to decision block 1815. This decision block determines whether the pattern to be removed has a one in the bit position of its MASK field indexed by the pivot bit of the pointed-to node, i.e., whether a wildcard exists in the associated stored pattern at this bit position (in which case the corresponding MASK bit would be zero).

Detailed Description Text (177):
If the indexed MASK bit in the pattern to be removed is zero, then the current branch node does not exist merely to support the pattern to be removed; therefore, the current branch node will remain in the structure. In this case, decision block 1815 routes execution, via NO path 1817, to block 1835. This latter block, when executed, recursively invokes routine 1800, i.e., Node Remove routine 1800, twice in succession: once for one child of the current pointed-to node and the other for the other child of that node. Once all such recursive executions have completed, block 1835 terminates its execution with execution passing to block 1840. This latter block updates the VALUE, MASK and IMASK fields of the pointed-to node. Specifically, the VALUE and MASK fields of the pointed-to node are set equal to the disjunction of the VALUE and MASK fields of both of its child nodes. The IMASK

field of the pointed-to node is set equal to the conjunction of IMASK fields of both of its child nodes. Once this occurs, execution exits from routine 1800.

Detailed Description Text (178):
However, if decision block 1815 determines that the pattern to be removed has a one-valued MASK bit at the bit position indexed by the pivot bit of the pointed-to node, i.e., a wildcard does not exist at this bit position in the pattern, then decision block 1815 routes execution, via YES path 1819, to block 1820. Hence, the search for the pattern to be removed can be confined to one child branch of the current node. Consequently, block 1820, when executed, recursively invokes this routine, i.e., Node Remove routine 1800, with, as an argument, the child node selected by a bit of the VALUE field of the pattern to be removed and indexed by the pivot bit in the pointed-to node. Once this recursive execution has fully completed to remove appropriate structure on one side of the pointed-to node, execution proceeds to decision block 1825. Through a two-part test, this decision block determines whether the current pointed-to branch node itself can be removed. In particular, block 1825 first determines whether, given the value of a bit of the VALUE field of the pattern to be removed indexed by the pivot bit, the pointed-to branch node has a corresponding child node. Second, this block determines whether a bit in the MASK field of that particular child node, at a bit position indexed by the pivot bit of the pointed-to node, is a one, i.e., whether any pattern node reachable through this branch node has a pattern that cares about the value of this particular bit position therein. If both tests are satisfied, then the current pointed-to branch node is necessary and can not be removed from the rhizome. In this case, decision block 1825 routes execution, via YES path 1827, to block 1840 to appropriately update, as described above, the fields associated with the current pointed-to branch node; after which, execution exits from routine 1800. Alternatively, if the current pointed-to branch node does not satisfy both of the conditions tested by decision block 1825, then the current pointed-to branch node is unnecessary and can be removed in its entirety without any adverse affects on the remaining rhizome structure. In this case, decision block 1825 routes execution, via NO path 1829, to block 1830. This latter block, when executed, replaces the current pointed-to branch node by one of its child nodes other than the corresponding child node selected within decision block 1825. Thereafter, the child node selected by decision block 1825 is eliminated. Finally, the current pointed-to branch node is de-allocated. Once block 1830 has fully executed, execution then exits from routine 1800.

Detailed Description Text (183):
Asymptotically, average search time in the rhizome is a logarithmic function of a number of elements stored in the rhizome rather than linear in the number of elements, as occurs with conventional databases of patterns containing wildcards. Hence, use of our inventive rhizome, particularly with a large classification database--as frequently occurs in packet classifiers, appears to provide significant time savings over conventional classification techniques known in the art.

Previous Doc Next Doc Go to Doc#